The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence
Planning for Hybrid Systems: Technical Report WS-16-12

A Compilation of the Full PDDL+ Language into SMT

Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni
Kings College London, London, WC2R 2LS
firstname.lastname@kcl.ac.uk

Abstract

Planning in hybrid systems is important for dealing with real-world applications. PDDL+ supports this representation of domains with mixed discrete and continuous dynamics, and supports *events* and *processes* modeling exogenous change. Motivated by numerous SAT-based planning approaches, we propose an approach to PDDL+ planning through SMT, describing an SMT encoding that captures all the features of the PDDL+ problem as published by Fox and Long (2006). The encoding can be applied on domains with nonlinear continuous change. We apply this encoding in a simple planning algorithm, demonstrating excellent results on a set of benchmark problems.

1 Introduction

PDDL+ planning is a growing area in the planning community, mainly motivated by the need to deal with real-world applications. PDDL+ (Fox and Long 2006) is the extension of PDDL designed to model hybrid dynamics, which are featured by a number of applications of interest for planning, and PDDL+ is pushing forward the use of planning in real-world domains (e.g., (Campion et al. 2013; Fox, Long, and Magazzeni 2011)). A number of approaches have been proposed that can handle *subsets* of PDDL+. Notoriously, most planners cannot handle events, or are limited to linear process models, as described later.

In this paper we propose a new approach for PDDL+ planning that can handle the whole set of PDDL+ features and respects Fox and Long's semantics. Our work builds on previous SAT encodings of planning problems (Kautz and Selman 1996; Rintanen 2010). We propose an SMT encoding of PDDL+ domains.

Planning through SMT is not new, although the approach presented here makes important contributions to PDDL+ planning through SMT. A recent approach, dReach, described in (Bryce et al. 2015) uses a non-linear SMT solver for planning in hybrid systems. That approach is promising, although it suffers some important limitations. Firstly, it does not use PDDL+, as it relies on the language of dReach, in which the hybrid problems have to be manually encoded. Secondly, dReach can only handle a restricted subset of the language features contained in PDDL+, and, in particular, they cannot handle events, which are a significant feature of PDDL+. Thirdly, dReach is tailored to be used only with the dReal solver (Gao, Avigad, and Clarke 2012).

In this paper, we propose a new encoding for PDDL+ which overcomes these limitations, and gives much better results in all the tested domains. In particular, our encoding is able to capture all features of PDDL+ (including events) and works by directly translating standard PDDL+ domain and problem files. Furthermore, we are careful to correctly capture the *must* semantics of PDDL+ (which constrains how processes and events interact with each other and with actions). Also, we model the precise semantics of ε-separation of effects and action preconditions (Fox and Long 2006). The output of the translation is a standard SMT encoding that can be used with any SMT solver in the theory of quantifier-free nonlinear arithmetic (QF_NRA). Furthermore, our approach proves to also be efficient in proving plan-non-existence, along with dramatically improving over dReach in solvable problems.

In terms of the dynamics our approach can handle, we can deal with nonlinear polynomial change. Moreover, when an instantaneous event triggers another, this can cause a cascade of events which all trigger simultaneously. Following the PDDL+ semantics (Fox and Long 2006), we assume to have a bound on the length of a causal chain of instantaneous events.

In the following we first introduce hybrid domains and PDDL+. We then present the new encoding, and use a working example to show how PDDL+ features are encoded in SMT. The overall approach is evaluated on a set of benchmark problems, and compared with previous works.

2 Hybrid Systems and PDDL+

A hybrid system is one in which there are both continuous control parameters and discrete logical modes of operation. It represents a powerful model to describe the dynamic behaviour of modern engineering artefacts. Hybrid systems frequently occur in practice, e.g., in robotics or embedded systems. Dealing with hybrid systems is becoming more and more an important challenge, as many real-world scenarios feature a mixture of discrete and continuous behaviours. Some example applications include coordination of activities of a planetary lander, oil refinery management, autonomous vehicles, chemical plant (Della Penna et al.

2010), smart grid (Campion et al. 2013), and battery management (Fox, Long, and Magazzeni 2011). Such scenarios motivate the need to reason with mixed discrete-continuous domains.

The theory of hybrid automata, introduced by Henzinger (Henzinger 1996), represents a well-defined formalism for describing hybrid systems.

2.1 Hybrid Automata

Intuitively, hybrid automata (Henzinger 1996) are finite state automata extended with continuous variables that evolve over time. More formally, we have the following:

Definition 1 (Hybrid Automaton). A *hybrid automaton* is a tuple $\mathcal{H} = (Loc, Var, Init, Flow, Trans, I)$, where

- Loc is a finite set of locations, $Var = \{x_1, \ldots, x_n\}$ is a set of real-valued variables, $Init(\ell) \subseteq \mathbb{R}^n$ is the set of initial values for $x_1, \ldots, x_n$ for all locations ℓ.
- For each location ℓ, $Flow(\ell)$ is a relation over the variables in Var and their derivatives of the form

$$\dot{x}(t) = Ax(t) + u(t), u(t) \in \mathcal{U},$$

  where $x(t) \in \mathbb{R}^n$, A is a real-valued $n \times n$ matrix and $\mathcal{U} \subseteq \mathbb{R}^n$ is a closed and bounded convex set.
- $Trans$ is a set of discrete transitions. A discrete transition $t \in Trans$ is defined as a tuple (ℓ, g, ξ, ℓ') where ℓ and ℓ' are the source and the target locations, respectively, g is the guard of t (given as a linear constraint), and ξ is the update of t (given by an affine mapping).
- $I(\ell) \subseteq \mathbb{R}^n$ is an invariant for all locations ℓ. □

An example is the hybrid automaton for a thermostat depicted in Figure 1. Here, the temperature is represented by the continuous variable x. In the discrete location corresponding to the heater being off, the temperature falls according to the flow condition $\dot{x} = -0.1x$, while, when the heater is on, the temperature increases according to the flow condition $\dot{x} = 5 - 0.1x$. The discrete transitions state that the heater *may* be switched on when the temperature falls below 19 degrees, and switched off when the temperature is greater than 21 degrees. Finally, the invariants state that the heater can be on (off) *only* if the temperature is not greater than 22 degrees (not less than 18 degrees).

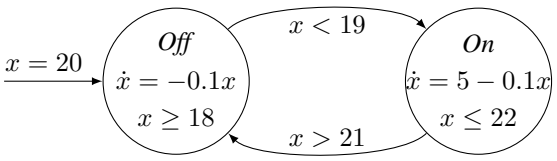

Figure 1: Thermostat hybrid automaton

2.2 PDDL+ Planning

PDDL+ is the extension of PDDL that allows modelling of mixed discrete-continuous domains, through continuous processes and events. PDDL+ is based on hybrid automata semantics, although there are some key difference, such as the *must* semantics. Continuous processes are triggered as soon as their precondition becomes true, and in this sense they *must* be triggered. Exogenous events follow the same semantics. The rationale behind this is that processes and events are used to model changes that are initiated by changing in the world, therefore they are not under the control of the executive and are triggered immediately (see (Bogomolov et al. 2014)) for more details).

Various techniques and tools have been proposed to deal with hybrid domains (Penberthy and Weld 1994; McDermott 2003; Li and Williams 2008; Coles et al. 2012; Shin and Davis 2005). More recent approaches in this direction have been proposed by (Bogomolov et al. 2014), where the close relationship between hybrid planning domains and hybrid automata is explored, and (Bryce et al. 2015) where hybrid domains are handled using SMT.

Nevertheless, none of these approaches are able to handle the full set of PDDL+ features, namely nonlinear domains with processes and events.

On the other hand, many works have been proposed in the model checking and control communities to handle hybrid systems. Some examples include (Cimatti et al. 2015; Cavada et al. 2014; Cimatti, Mover, and Tonetta 2012; Tabuada, Pappas, and Lima 2002; Maly et al. 2013), sampling-based planners (Karaman et al. 2011; Lahijanian, Kavraki, and Vardi 2014). Another related direction is *falsification* of hybrid systems (i.e., guiding the search towards the error states, that can be easily cast as a planning problem) (Plaku, Kavraki, and Vardi 2013). However, while all these works aim to address a similar problem, that cannot be used to handle PDDL+ models. Some recent works (Bogomolov et al. 2014; 2015) are trying to define a formal translation between PDDL+ and standard hybrid automata, but so far only an over-approximation has been defined, that allows the use of those tools only for proving plan non-existence.

To date, the only viable approach in this direction is PDDL+ planning via discretisation. UPMurphi (Della Penna, Magazzeni, and Mercorio 2012), which implements the discretise and validate approach, is able to deal with the full range of PDDL+ features.

3 PDDL+ as SMT

PDDL+ supports the representation of domains with mixed discrete-continuous dynamics, providing a flexible model of continuous change. In this section we will describe the PDDL+ planning model, as an extention of PDDL2.1. Then, we will describe our encoding of a PDDL+ planning problem as an SMT formula.

Definition 2 (PDDL2.1 Planning Problem). A PDDL2.1 planning problem is a tuple $\Pi := \{P, V, A, I, G\}$, where P is a set of propositions; V is a vector of real variables, called fluents; both are manipulated by A, a set of durative and instantaneous actions. $I(P, V)$ is a fuction over $P \cup V$ which describes the *initial state* of the problem. Similarly $G(P, V)$ is a function that describes the *goal condition*. □

A durative action a is described as a tuple:

$$a := \{pre_a, eff_a, dur_a\}$$

where pre_a represents the action's preconditions – conditions that must hold for the action to be applied – eff_a represents the action's effects, and dur_a is a duration constraint, a conjunction of numeric constraints corresponding to the duration of the action a.

A single condition is either a single proposition $p \in P$, or a numeric constraint over V. A precondition is a conjunction of zero or more conditions. Each durative action A has three disjoint subsets of preconditions:

$$pre_{\vdash a}, pre_{\leftrightarrow a}, pre_{\dashv a} \subseteq pre_a$$

These represent the conditions that must hold at its start, throughout its execution, and at the end of the action, respectively.

Action effects are described by seven subsets:

$$\begin{array}{l} \mathit{eff}^{+}_{\vdash a},\ \mathit{eff}^{-}_{\vdash a},\ \mathit{eff}^{num}_{\vdash a}, \\ \mathit{eff}^{+}_{\dashv a},\ \mathit{eff}^{-}_{\dashv a},\ \mathit{eff}^{num}_{\dashv a}, \\ \mathit{eff}_{\leftrightarrow a} \end{array}$$

$\mathit{eff}_{\leftrightarrow a}$ is a conjunction of *continuous* numeric effects $\mathit{eff}_{\leftrightarrow}$, which are applied continuously while the action is executing. The rest are instantaneous effects, adding or removing propositions, or numeric effects. These are bound to the start or end of the action. For example, $\mathit{eff}^{+}_{\vdash a}$ denotes the propositions added at the start of the action. Semantically, the values of such instantaneous effects can be exploited to support actions only after a small of time ε (Fox and Long 2003).

As a special case, *instantaneous* actions have duration 0, have only one set of preconditions pre_a; and three sets of effects eff^{+}_{a}, eff^{-}_{a}, and eff^{num}_{a}.

PDDL+ extends PDDL2.1 to support the modelling of exogenous events, reflecting changes that are initiated by the environment. PDDL+ introduces the new constructs of *processes* and *events*.

As an analogue, events are akin to instantaneous actions: if an event's preconditions pre_a are satisfied, it occurs, yielding the event's instantaneous effects. Similarly, processes are akin to durative actions. The critical distinctions between processes and events, and actions, is that a process/event will automatically occur as soon as its precondition is satisfied; whereas an action will only happen if chosen to be executed in the plan. Furthermore, that the values of events become available instantaneously. Therefore, if one event $e1$ is triggered, with effects that satisfy another event $e2$ and trigger a process $p1$, then $e1$, $e2$, and the start of $p1$ all happen at the same time-point. It is due to this behaviour that we place a bound on the number of cascading (parallel) events.

Definition 3 (PDDL+ Planning Problem). A PDDL+ planning problem is a tuple $\Pi+ := \langle P, V, A, Ps, E, I, G \rangle$, in which P is a set of propositions; V is a vector of real variables, called fluents; and A is a set of durative and instantaneous actions. Ps is a set of processes, and E a set of events. $I(P, V)$ and $G(P, V)$ represent the initial state and goal condition respectively. □

3.1 Encoding of PDDL+ Domains

In this section we describe how we encode a PDDL+ domain. First, we introduce the notion of *happening*, that is used to capture the change in the state at a given time point, due to the effects of actions, processes or events happening at that time point. Namely, each happening encodes the causal chain of processes, events and instantaneous actions which might occur simultaneously at a given time point. Note that, according to the PDDL+ semantics, effects of actions apply ε time after they occur, therefore we need to consider the change happening in the time intervals $t + \varepsilon$.

An example of happening is shown in Figure 2.

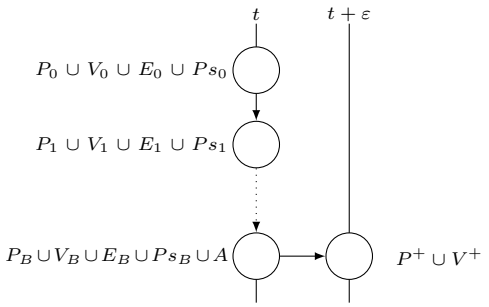

Figure 2: A single happening, with a bound of B cascading events. The happening occurs at time t, at which there are several sets of state variables. These sets describe a causal chain of instantaneous events and processes triggers. Actions are included at the end of this chain.

Formally, we have the following

Definition 4 (Happening). A happening is the tuple $x := \{t, \hat{P}, \hat{V}, \hat{Ps}, \hat{E}, \hat{A}, P^+, V^+\}$, where:

- t is the current time point;
- $\hat{P} = \{P_0, \ldots, P_B\}$ represents the causal change in the propositional state variables P at time t;
- $\hat{V} = \{V_0, \ldots, V_B\}$ represents the causal change in the real state variables V at time t;
- $\hat{Ps} = \{Ps_0, \ldots, Ps_B\}$ represents the chain of active processes Ps at time t;
- $\hat{E} = \{E_0, \ldots, E_B\}$ represents the chain of events triggered at time time t;
- $\hat{A}$ is the set of actions applied at time t.
- P^+ is the value of propositional state variables at time $t + \varepsilon$;
- V^+ is the value of real state variables at time $t + \varepsilon$. □

For our encoding we split durative actions as two intantaneous actions, representing the start and end of the action, and one process representing the continuous numerical effects and invariant.

A happening describes a moment of discrete change, corresponding to the discrete transition *Trans* of the Hybrid Automata. Between happenings there is only continuous numeric change (*Flow*). The key differences are that: multiple actions can be performed in a single happening in parallel, meaning that while the hybrid automata is exponential in the size of the PDDL+ description, our encoding will be linear;

and the continuous change is not limited to first order derivatives.

Having defined happenings, a PDDL+ model can be described as a bounded set of happenings $X := \{x_1...x_n\}$ encoded as an SMT formula, such that any proof for the SMT formula represents the trace of a valid plan for $\Pi+$. The *plan* corresponding to that trace is the set of action assignments $(\hat{A})_1 \cup ... \cup (\hat{A})_n$.

We first describe the encoding of a single happening, and then describe the encoding of the formula for $\Pi+$.

Encoding of a Happening Following from Definition 4, we encode a happening x as follows:

$$x := \left\langle \begin{array}{c} t,\ P_0, ..., P_B,\ V_0, ..., V_B, \\ E_0, ..., E_B,\ Ps_0, ..., Ps_B, \\ A,\ P^+,\ V^+,\ flow_V,\ dur_{Ps} \end{array} \right\rangle$$

We recall that we assume a bound B on the length of the causal chain of events at each time point. If the causal chain is longer than this, then a valid plan will not be found.

Any actions applied at time point t are represented in the set A. The sets P^+ and V^+ describe the values of the state variables at time $(t+\varepsilon)$, containing the instantaneous effects of these actions, according to the PDDL+ semantics (Fox and Long 2006). Actions can only be applied together in the same happening if they are not mutually exclusive (mutex). Actions $a1$ and $a2$ can be applied simultaneously if:

$$pre_{a1} \cap (eff^+_{a2} \cup eff^-_{a2} \cup eff^{num}_{a2}) = \emptyset$$
$$pre_{a2} \cap (eff^+_{a1} \cup eff^-_{a1} \cup eff^{num}_{a1}) = \emptyset$$
$$eff^+_{a1} \cap eff^-_{a2} = eff^+_{a2} \cap eff^-_{a1} = \emptyset$$
$$\{v1 | \forall v1 \in eff^{num}_{a1}\} \cap \{v2 | \forall v2 \in eff^{num}_{a2}\} = \emptyset$$

The set $flow_V := \{flow_v | \forall v \in V\}$ is a numerical expression that represents the change in value of v from this time point to the next. Finally, $dur_{Ps} := \{dur_{ps} | \forall ps \in Ps\}$ represents the remaining duration of each process. dur_{ps} is constrained to be positive if and only if the process is currently executing.

The constraints within a happening are shown in Figure 3. *Proposition and real variable support* constraints ensure that the value of propositions (H1-H2) and real variables (H3) remains consistent from $P_0 \cup V_0$ to $P_B \cup V_B$. *Event preconditions and effects* constraints enforce that an event is triggered if and only if its precondition holds (H4) and that if an event is triggered, its effects are present in the next set (H5).

Action preconditions and effects ensure the same rules apply for actions in A; their preconditions must hold in $P_B \cup V_B$ (H6) and their effects are enforced in $P^+ \cup V^+$ (H7). *Support across epsilon separation* ensures that the value of propositions (H8-H9) and real variables (H10) remains consistent from $P_B \cup V_B$ to $P^+ \cup V^+$.

Process triggering constraints enforce that a process is active if and only if its preconditions are satisfied in each set $P_0 \cup V_0$ to $P_B \cup V_B$ (H11). It also enforces that the real variable dur_{ps} for each process is greater than or equal to zero (H12), and that if and only if the process is active at the end of the causal chain of events, then the duration is strictly greater than zero (H13). These constraints will be used to ensure that a process cannot finish outside of a happening.

Finally, *Action mutexes* are included as a collection of binary constraints, enforcing that no two mutex actions can be applied simultaneously. For each pair of mutex actions, denoted $a \nparallel a'$, one must be false (H14).

Encoding of a Planning Problem $\Pi+$ The existence of a plan for a *PDDL+ planning problem* $\Pi+$ *with bound* n is proved by building the SMT formula $\Pi+_n$ in the theory of quantifier-free (nonlinear) real arithmetic with n copies of the set of variables x $(x_1...x_n)$ for $n \geq 1$. A *plan for* $\Pi+$ is the assignment to the action variables in any proof of $\Pi+_n$. The encoding is illustrated by Figure 4.

The constraints for a happening in Figure 3 are copied for each happening $x_0...x_n$. Additional constraints in the SMT formula $\Pi+_n$ are shown in Figure 5 and explained below.

The *Instance description* constraints enforce the initial state to hold in the first happening (P1), and that the goal is achieved in the final happening (P2). Also, they constrain the timing of happenings to enforce epsilon separation (P3-P4). *Proposition support* simply ensures that the discrete state variables P do not change between two happenings (P5-P6).

Invariant constraints ensure that the continuous numeric change between happenings is valid. This is achieved in three parts. First (P7) ensures that if a process is active in the previous happening, its duration is decreased by the time between happenings. This constraint, in combination with constraints (P12-P13) of figure 3, ensure that a process cannot end between happenings. Note that a process can remain active over intervals spanning multiple happenings.

Constraint (P8) enforces the invariant of the process. If a process is active, then the precondition of the process is active over the whole interval between happenings, and if the process is not active, then the precondition is false over the whole interval. For constraints over real valued variables, this is done by checking the value either side of the interval. For nonlinear change, it is necessary to include the following additional constraint:

$$ps_i \rightarrow \bigwedge_{a=1}^{A} \left((\frac{d^a f}{dt^a})_i (\frac{d^a f}{dt^a})_{i+1} >= 0 \right)$$

where f is the numeric, non-constant part of the invariant, and where the $(A+1)$th derivative of f is identically zero. This ensures that the derivatives of the function do not cross zero over the interval, thus a fluctuating value of f cannot violate the invariant condition between i and $i+1$.

Constraint (P9) similarly ensures that an event is not triggered during an interval. Here $\neg(pre_{\leftrightarrow e})$ is the condition that pre_e does not hold over the whole interval.

Continuous change on real variables is enforced by calculating the change over the interval (P10) and applying it to each real variable (P11). In order to calculate the change, the indefinite integral of the process' effects upon the variable must be computed. This is done automatically using the computer algebra system (CAS) SymPy (SymPy 2013).

Note that in our approach, the integration and differentiation required for P8, P9, and P10 is performed by SymPy

Proposition and real variable support

H1. $\bigwedge_{i=0}^{B-1} \bigwedge_{p \in P} p_{i+1} \rightarrow (p_i \bigvee_{e|p \in eff_e^+} e_i)$

H2. $\bigwedge_{i=0}^{B-1} \bigwedge_{p \in P} \neg p_{i+1} \rightarrow (\neg p_i \bigvee_{e|p \in eff_e^-} e_i)$

H3. $\bigwedge_{i=0}^{B-1} \bigwedge_{v \in V} (\bigwedge_{e|v \in eff_e^{num}} \neg v_i) \rightarrow (v_{i+1} = v_i)$

Event preconditions and effects

H4. $\bigwedge_{i=0}^{B} \bigwedge_{e \in E} e_i \leftrightarrow (pre_e)_i$

H5. $\bigwedge_{i=0}^{B-1} \bigwedge_{e \in E} e_i \rightarrow (eff_e)_{i+1}$

Action preconditions and effects

H6. $\bigwedge_{a \in A} a \rightarrow (pre_a)_B$

H7. $\bigwedge_{a \in A} a \rightarrow (eff_a)^+$

Support across epsilon separation

H8. $\bigwedge_{p \in P} p^+ \rightarrow (p_B \bigvee_{a|p \in eff_a^+} a)$

H9. $\bigwedge_{p \in P} \neg p^+ \rightarrow (\neg p_B \bigvee_{a|p \in eff_a^-} a)$

H10. $\bigwedge_{v \in V} (\bigwedge_{a|v \in eff_a^{num}} \neg a) \rightarrow (v^+ = v_B)$

Process triggering

H11. $\bigwedge_{i=0}^{B} \bigwedge_{ps \in Ps} ps_i \leftrightarrow (pre_{ps})_i$

H12. $\bigwedge_{ps \in Ps} dur_{ps} >= 0$

H13. $\bigwedge_{ps \in Ps} ps_B \leftrightarrow (dur_{ps} > 0)$

Action mutexes

H14. $\bigwedge_{a \in A} \bigwedge_{a' \in A|a \nparallel a'} (\neg a \vee \neg a')$

Figure 3: Reduction of a PDDL+ happening to SMT.

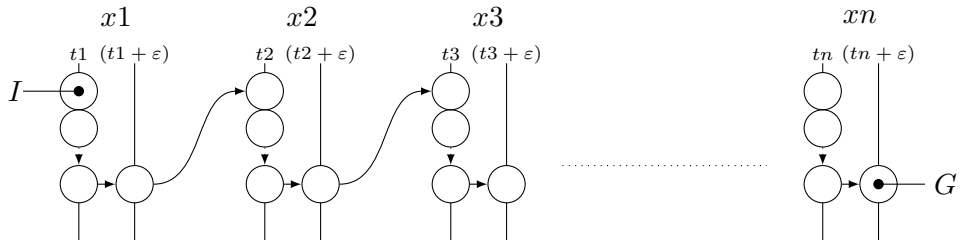

Figure 4: A plan is found by building a formula with n copies of the set of variables x: $x_1 ... x_n$. Each Happening models discrete change. Between happenings there is only continuous numerical change. The initial state is modelled in x_1, and the goal constraints are added to x_n.

Instance description

P1. $I((P_0)_1 \cup (V_0)_1)$

P2. $G((P^+)_n \cup (V^+)_n)$

P3. $t_1 = 0$

P4. $\bigwedge_{i=2}^{n} t_i \geq t_{i-1} + \varepsilon$

Proposition support

P5. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} (p_0)_i \rightarrow (p^+)_{i-1}$

P6. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} \neg (p_0)_i \rightarrow \neg (p^+)_{i-1}$

Invariants

P7. $\bigwedge_{i=2}^{n} \bigwedge_{ps \in Ps} (ps_B)_{i-1} \rightarrow ((dur_{ps})_i = (dur_{ps})_{i-1} + t_i - t_{i+1})$

P8. $\bigwedge_{i=1}^{n-1} \bigwedge_{ps \in Ps} (ps_B)_i \leftrightarrow (pre_{\leftrightarrow ps})_i$

P9. $\bigwedge_{i=1}^{n-1} \bigwedge_{e \in E} \neg (pre_{\leftrightarrow e})_i$

Continuous change on real variables

P10. $\bigwedge_{i=1}^{n-1} \bigwedge_{v \in V} (flow_v)_i = \int_{t_i}^{t_{i+1}} \cup_{ps \in Ps} (eff_{\leftrightarrow ps}^{num}[v])_i dt$

P11. $\bigwedge_{i=2}^{n} \bigwedge_{v \in V} ((v_0)_i = (v^+)_{i-1} + (flow_v)_{i-1})$

Figure 5: Reduction of PDDL+ planning problem $\Pi+$ to SMT.

outside the solver, during the encoding. Hence the integration is done only once for each domain.

4 Example: Simple Generator

We will walk through the encoding of a simple PDDL+ problem, the *simple generator* problem. The domain is shown in Figure 6. The initial state asserts that there is one generator; the generator's capacity C; and initial fuel level F. The goal state is that the generator has been run: `(generator-ran)`.

We will show the encoding of three happenings x_1 and x_2 – the minimum number of steps required to solve this problem. As there are no events in the domain, we will impose a bound $B = 0$ on the number of cascading events. Therefore each happening is the set:

$$x_i := \left\langle \begin{array}{ll} gen\_ran_0, refueling_0 & : Bool \\ fuelLevel_0, capacity_0 & : Real \\ gen\_ran^+, refueling^+ & : Bool \\ fuelLevel^+, capacity^+ & : Real \\ gen\_start, gen\_end, gen\_process & : Bool \\ dur\_gen\_process & : Real \\ flow\_fuelLevel, flow\_capacity & : Real \end{array} \right\rangle$$

Additionally, we include the variables: $t_1, t_2 : Real$, which describe the time point of each happening. The resultant plan is shown by assignment to variables in Table 1.

```
(define (domain simple_generator)
(:requirements
    :fluents :durative-actions
    :duration-inequalities :adl
    :typing)
(:types generator)
(:predicates
    (generator-ran)
    (refueling ?g - generator)

(:functions
    (fuelLevel ?g - generator)
    (capacity ?g - generator))

(:durative-action generate
 :parameters (?g - generator)
 :duration (= ?duration 1000)
 :condition (over all (>= (fuelLevel ?g) 0))
 :effect (and
    (decrease (fuelLevel ?g) (* #t 1))
    (at end (generator-ran)))))
```

Figure 6: Simplified PDDL+ generator domain

x_1	x_2
$t_1 := 0$	$t_2 := 1000$
$fuelLevel_0 = 1020.0$	$fuelLevel_0 = 20.0$
$fuelLevel^+ = 1020.0$	$fuelLevel^+ = 20.0$
$capacity_0 = 1060.0$	$capacity_0 = 1060.0$
$capacity^+ = 1060.0$	$capacity^+ = 1060.0$
gen_start	$refueling_0$
$gen\_process$	gen_end
$refueling^+$	gen_ran^+
$dur\_gen\_process = 1000.0$	$dur\_gen\_process = 0.0$
$flow\_fuelLevel = -1000.0$	$flow\_fuelLevel = 0.0$
$flow\_capacity = 0$	$flow\_capacity = 0$

Table 1: A plan for the simple generator domain, as assignment to the variables. Boolean variables are shown in the table if and only if they are assigned true.

The formula for $\Pi+$ consists of the following constraints in conjunctive normal form (CNF): the instance description enforces the initial state, the goal condition, and that that the second happening is at least ε later than the first.

$$
\begin{array}{l}
(0 = t_1) \\
\neg(gen\_ran_0)_1 \\
\neg(refueling_0)_1 \\
(F = (fuelLevel_0)_1) \\
(C = (capacity_0)_1) \\
(gen\_ran^+)_2 \\
(t_2 >= (t_1 + \varepsilon))
\end{array}
$$

Proposition support constraints (between happenings) ensure that discrete state stays constant during the interval[1]:

$$
\begin{array}{l}
(gen\_ran^+)_1 = (gen\_ran_0)_2 \\
(refueling^+)_1 = (refueling_0)_2
\end{array}
$$

[1]All the formulae in the following are represented in conjunctive normal form (CNF).

Invariant constraints ensure that the durative action's overall condition holds over an interval in which its associated process is active, and that the durative action's duration is properly updated across the interval:

$$
\begin{array}{l}
(gen\_process)_1 \rightarrow ((fuelLevel^+)_1 >= 0) \\
(gen\_process)_1 \rightarrow ((fuelLevel_0)_2 >= 0) \\
(gen\_process)_1 \rightarrow \\
\quad ((dur\_gen\_process)_2 = \\
\quad\quad ((dur\_gen\_process)_1 + t_1 - t_2))
\end{array}
$$

The continuous change over real variables is defined and enforced by the $flow$ variables:

$$
\begin{array}{l}
(gen\_process)_1 \rightarrow (flow\_fuelLevel)_1 = \int_{t1}^{t2}(-1.0)dt \\
\neg(gen\_process)_1 \rightarrow (flow\_fuelLevel)_1 = 0 \\
(flow\_capacity)_1 = 0 \\
(fuelLevel_0)_2 = (fuelLevel^+)_1 + (flow\_fuelLevel)_1 \\
(capacity_0)_2 = (capacity^+)_1 + (flow\_capacity)_1
\end{array}
$$

Proposition support constraints (within happenings) and action preconditions and effects, describe the discrete changes that occurs within a happening, for $i = \{1, 2\}$:

$$
\begin{array}{l}
(gen\_ran^+)_i \rightarrow (gen\_ran_0)_i \vee (gen\_end)_i \\
\neg(gen\_ran^+)_i \rightarrow \neg(gen\_ran_0)_i \\
(refueling_0)_i = (refueling^+)_i \\
(fuelLevel_0)_i = (fuelLevel^+)_i \\
(capacity_0)_i = (capacity^+)_i \\
(gen\_start)_i \rightarrow (gen\_process)_i \\
(gen\_end)_i \rightarrow (gen\_ran^+)_i \\
(gen\_end)_1 \leftrightarrow (gen\_process)_1 \wedge \neg(gen\_process)_2
\end{array}
$$

Process triggering constraints work together to ensure that the duration of the durative action is within the constraints of the durative action, and that it begins and ends within the happenings:

$$
\begin{array}{l}
(dur\_gen\_process)_1 >= 0 \\
\neg(gen\_process)_1 \leftrightarrow ((dur\_gen\_process)_1 = 0) \\
(gen\_start)_1 \leftarrow ((dur\_gen\_process)_1 = 1000.0) \\
\neg(gen\_start)_1 \leftarrow ((dur\_gen\_process)_1 = 0.0)
\end{array}
$$

Finally, action mutexes are included. In our encoding, we make the starts and ends of durative actions mutually exclusive, eg. for $i = \{1, 2\}$:

$$
\neg(gen\_start)_i \vee \neg(gen\_end)_i
$$

5 Results

We use our encoding to solve PDDL+ planning problems with a parallel iterative deepening technique, widely used in SAT-based planning approaches (Nabeshima, Iwanuma, and Inoue 2002; Rintanen, Heljanko, and Niemel 2006). The top-level algorithm encodes and solves n SMT encodings simultaneously, solving the planning problem for horizon lengths $1, 2, 3, 4, ..., n$. In this case, horizon length corresponds to number of happenings. If a formula is found satisfiable, then a plan has been found and the planner terminates. If a formula is found unsatisfiable, then an encoding is made for the next shortest horizon length, so that there are always

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan+	0.02	0.03	0.02	0.01	0.02	0.02	0.02	0.02
	dReach	2.87	-	-	-	-	-	-	-
	UPMurphi	0.2	18.2	402.34	-	-	-	-	-
Generator nonlinear	SMTPlan+	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
	dReach	5.16	-	-	-	-	-	-	-
	UPMurphi	-	-	-	-	-	-	-	-
Generator nonlin. events	SMTPlan+	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.05
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	658.18	-	-	-	-	-	-	-
Generator Torricelli	SMTPlan+	0.03	0.03	0.15	0.92	0.04	0.05	0.09	0.50
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	-	-	-	-	-	-	-	-
Car	SMTPlan+	0.02	0.02	0.02	0.02	0.02	0.02	0.01	0.02
	dReach	1.30	1.41	1.48	1.53	1.47	1.54	1.40	1.53
	UPMurphi	28.44	386.5	-	-	-	-	-	-

Table 2: Results in seconds for solvable instances. Instance numbers correspond to number of tanks (generator) and number of acceleration steps (car). Abbrev.: -: tool still running after 30 minutes, '.': tool ran out or memory, x: tool cannot handle the problem.

n SMT instances being solved. The SMT solver we use is z3 (De Moura and Bjørner 2008).

We compare our approach (called SMTPlan+) against existing PDDL+ planner UPMurphi (Della Penna, Magazzeni, and Mercorio 2012), and with dReach (Bryce et al. 2015), using the SMT solver dReal (Gao, Avigad, and Clarke 2012), on domains without events. We use the *generator* and *car* domains (Bogomolov et al. 2014). The experiments were run using 8GB of RAM and a 30 minute timeout. All test domains, problems and plans are available at: *[Website omitted for author anonymity]*.

Generator The generator domain is a PDDL+ benchmark problem that revolves around refueling a diesel-powered generator which has to run for a given duration without overflowing or running dry. To test scalability, the number of tanks is increased while decreasing the initial generator fuel level.

We consider four versions of this domain: linear, simplified-nonlinear (the same used in (Bryce et al. 2015)), nonlinear with events, and the Torricelli nonlinear (Howey and Long 2003). Note that the latter version uses the Torricelli's Law (which is too complex for dReach), and hence the fuel level in a refueling tank (V_{fuel}) is calculated by:

$$V_{fuel} = (-kt_r + \sqrt{V_{init}})^2 \qquad t_r \in \left[0, \frac{\sqrt{V_{init}}}{k}\right] \qquad (1)$$

where V_{init} is the initial volume of fuel in the tank, k is the fuel flow constant (which depends on gravity, size of the drain hole, and the cross-section of the tank), and t_r is the time of refueling (bounded by the fuel level and the flow constant).

Here is an example of plan found by SMTPlan+ for the Torricelli nonlinear generator (Fuel level 960, Generator capacity 990):

```
  0.0: generate     [1000.0]
959.0: refuel_tank1 [10.0]
959.0: refuel_tank2 [10.0]
```

Car The car domain is another PDDL+ benchmark (Fox and Long 2006) where a vehicle has to cover a given distance and have a zero velocity at the end, and the actions available are accelerate and decelerate that increments or decrements by 1 the current velocity, respectively. To test scalability, the bound on maximum acceleration/deceleration is increased.

Our results for solvable instances are reported in table 2. On both linear and nonlinear domains, SMTPlan+ outperforms all other planners in time to solve and in number of instances solved. In all domains, SMTPlan+ scales very well. For these domains, the number of happenings required is small, thus the minimal SMT encoding required to solve the problem is also small. The iterative deepening algorithm is able to reach a satisfiable encoding, and produce a plan very quickly.

dReach also performs iterative deepening, but performs more poorly. This is due to the semantics of dReach; in the dReach domain and problem description, each mode of continuous change must be explicitly defined, and the number of modes increases exponentially with the number of processes and durative actions (eg. the files for 1, 2, 3 and 4 tanks problems are respectively 91, 328, 1350, 5762 lines long). Furthermore, the bound is not on the number of happenings, but on the number of mode changes, which does not allow for parallel execution of actions.

Moreover, dReach does not perform integration and differentiation outside of the solver, during encoding. Instead it relies upon the more expressive logic of the internal SMT solver, dReal, at the cost of extra computation time. In addition, they are unable to use SMT solvers other than dReal.

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan+	0.01	0.02	0.16	2.84	390.86	-	-	-
	dReach	2.57	189.94	-	-	-	-	-	-
	UPMurphi	0.90	29.42	-	-	-	-	-	-
Generator nonlinear	SMTPlan+	0.01	1.95	33.48	-	-	-	-	-
	dReach	2.43	212.43	-	-	-	-	-	-
	UPMurphi	-	-	-	-	-	-	-	-
Generator nonlin. events	SMTPlan+	0.02	18.58	21.83	-	-	-	-	-
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	658.18	-	-	-	-	-	-	-
Generator Toricelli	SMTPlan+	0.03	2.06	19.57	-	-	-	-	-
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	-	-	-	-	-	-	-	-
Car	SMTPlan+	0.68	0.02	0.00	0.00	0.00	0.00	0.00	0.01
	dReach	0.67	0.50	0.62	0.45	0.58	0.57	0.49	0.65
	UPMurphi	36.01	445.23	-	-	-	-	-	-

Table 3: Results in seconds for unsolvable instances. Instance numbers correspond to number of tanks (generator) and number of acceleration steps (car). Abbrev.: -: tool still running after 30 minutes, x: tool cannot handle the problem.

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan+	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02
	dReach	2.73	13.47	104.61	695.70	-	-	-	-
Generator nonlinear	SMTPlan+	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	dReach	10.42	1685.35	-	-	-	-	-	-
Car	SMTPlan+	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	dReach	0.77	0.76	0.76	0.76	0.76	0.76	0.77	0.76

Table 4: Results in seconds for minimal step encoding required to solve each instance.

We also compare our encoding directly against dReach as reported in prior work (Bryce et al. 2015): reporting times to solve only the encoding of a minimal step plan for each instance. These are not the times required to solve a PDDL+ instance, but a direct comparison of encodings on satisfiable problems. We find the encodings exhibit similar performance in the car domain. However, we find the SMTPlan+ encoding scales far better on the generator problem, as discussed above. Moreover, the SMTPlan+ encoding does not require the advanced features of dReal, and can be solved more quickly using z3.

Our results for unsolvable instances are shown in table 3. SMTPlan+ and dReach can only prove unsolvability up to an upper bound on the number of happenings. Here we prove plan non-existence for domains which have a tight deadline, and where each ground action can only be applied a finite number of times. . We also include SpaceEx that can be used to prove plan-non existence for the generator linear domain (Bogomolov et al. 2014). We observe that both totally ordered planning approaches perform well proving unsolvability in the car domain. There are few choices of symbolic plan in this domain, leaving only the timing of the happenings and numeric constraints to be solved. Both SMTPlan+ and dReach solve these constraints very quickly. However, for PDDL+ problems in general, without deadlines and with repeatable actions, proving unsolvability is difficult through totally ordered planning with iterative deepening.

6 Conclusion

In this paper we presented a new approach for PDDL+ planning that can handle the whole set of PDDL+ features and respects Fox and Longs semantics. We proposed an SMT encoding of PDDL+ domains that correctly captures the must semantics of PDDL+ which constrains how processes and events interact with each other and with actions. The encoding is general and can be used with any SMT solver in the theory of quantifier-free nonlinear arithmetic.

Experimental results show that the approach dramatically outperforms existing work in finding plans for solvable problems, and it is efficient also in proving plan-non-existence.

References

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2228–2234.

Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *Proceedings of the*

Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS, 42–46.

Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. SMT-based nonlinear PDDL+ planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 3247–3253.

Campion, J.; Dent, C.; Fox, M.; Long, D.; and Magazzeni, D. 2013. Challenge: Modelling unit commitment as a planning problem. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS*.

Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; and Tonetta, S. 2014. The nuXmv symbolic model checker. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, 334–342.

Cimatti, A.; Griggio, A.; Mover, S.; and Tonetta, S. 2015. HyComp: An SMT-based model checker for hybrid systems. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, ETAPS*, 52–67.

Cimatti, A.; Mover, S.; and Tonetta, S. 2012. SMT-based verification of hybrid systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

De Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340.

Della Penna, G.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2010. A PDDL+ benchmark problem: The batch chemical plant. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS*, 222–225.

Della Penna, G.; Magazzeni, D.; and Mercorio, F. 2012. A universal planning system for hybrid domains. *Applied Intelligence* 36(4):932–959.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Res. (JAIR)* 20:61–124.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Jorunal of Artificial Intelligence Research (JAIR)* 27:235–297.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic construction of efficient multiple battery usage policies. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, 2620–2625.

Gao, S.; Avigad, J.; and Clarke, E. M. 2012. Delta-complete decision procedures for satisfiability over the reals. *CoRR*.

Henzinger, T. A. 1996. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, 278–292.

Howey, R., and Long, D. 2003. VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proc. of ICAPS Workshop on the IPC*.

Karaman, S.; Walter, M. R.; Perez, A.; Frazzoli, E.; and Teller, S. J. 2011. Anytime motion planning using the RRT. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, 1478–1483.

Kautz, H., and Selman, B. 1996. Pushing the envelope: planning, propositional logic and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, 1194–1201.

Lahijanian, M.; Kavraki, L. E.; and Vardi, M. Y. 2014. A sampling-based strategy planner for nondeterministic hybrid systems. In *IEEE International Conference on Robotics and Automation, ICRA*, 3005–3012.

Li, H. X., and Williams, B. C. 2008. Generative planning for hybrid systems based on flow tubes. In *ICAPS*, 206–213.

Maly, M. R.; Lahijanian, M.; Kavraki, L. E.; Kress-Gazit, H.; and Vardi, M. Y. 2013. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC*, 353–362.

McDermott, D. V. 2003. Reasoning about autonomous processes in an estimated-regression planner. In *ICAPS*, 143–152.

Nabeshima, H.; Iwanuma, K.; and Inoue, K. 2002. Effective sat planning by speculative computation. In *AI 2002: Advances in Artificial Intelligence*, volume 2557 of *Lecture Notes in Computer Science*. 726–726.

Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In *AAAI*, 1010–1015.

Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2013. Falsification of LTL safety properties in hybrid systems. *STTT* 15(4):305–320.

Rintanen, J.; Heljanko, K.; and Niemel, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170:1031–1080.

Rintanen, J. 2010. Madagascar: Efficient planning with SAT. In *The 7th International Planning Competition*, 61–64.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artif. Intell.* 166(1-2).

SymPy. 2013. Website. http://www.sympy.org/.

Tabuada, P.; Pappas, G. J.; and Lima, P. U. 2002. Composing abstractions of hybrid systems. In *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC*, 436–450.